A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems

Emanuel Falkenauer
CRIF - Research Centre for Belgian
Metalworking Industry
Industrial Automation Division
CP 106 - P4
50, av. F. D. Roosevelt
B-1050 Brussels
Belgium
fal@crif.ulb.ac.be

Abstract

An important class of computational problems are *grouping* problems, where the aim is to group together members of a set (i.e., find a good *partition* of the set). We show why both the standard and the ordering GAs fare poorly in this domain by pointing out their inherent difficulty to capture the regularities of the functional landscape of the grouping problems. We then propose a new encoding scheme and genetic operators adapted to these problems, yielding the *Grouping* Genetic Algorithm (GGA). We give an experimental comparison of the GGA with the other GAs applied to grouping problems, and we illustrate the approach with two more examples of important grouping problems successfully treated with the GGA: the problems of Bin Packing and Economies of Scale.

Keywords

Genetic algorithm, grouping, partitioning, solution encoding, genetic operators

1. Introduction—The Grouping Problems

Many problems naturally arising in practice consist in *partitioning* a set $\mathbf{U}$ of objects into a collection of mutually disjoint subsets $\mathbf{U}_i$ of $\mathbf{U}$, such that

$$\cup\, \mathbf{U}_i = \mathbf{U}$$

and

$$\mathbf{U}_i \cap \mathbf{U}_j = \varnothing,\ i \neq j.$$

One can also see these problems as those where the aim is to *group* the members of the set $\mathbf{U}$ into one or more (at most **card**($\mathbf{U}$)) groups of objects, with each object in exactly one group, i.e., to find a grouping of those objects.

In most of these problems, not all possible groupings are allowed: a solution to the problem must comply with various hard constraints, otherwise the solution is invalid. That is, usually an object cannot be grouped with all possible subsets of the remaining objects.

The objective of the grouping is to optimize a cost function defined over the set of all valid groupings. The following are just three examples[1] of well-known grouping problems, with the hard constraint to which a solution must comply, and the cost function to minimize:

Problem	*Hard Constraint*	*Cost Function*
Bin Packing	Sum of sizes of objects in any group < C	Number of groups
Workshop Layouting	Number of machines in any group < C	Total intercell traffic
Graph Coloring	No connected nodes in any group	Number of groups

As can be seen, the grouping problems are characterized by cost functions that depend on the *composition of the groups*[2], that is, where one object taken in isolation has little or no meaning.

Many grouping problems (including all the examples above) are NP-hard (see Garey & Johnson (1979)), which implies that any known exact algorithm will run in time exponential in the size of the problem instance. Such an algorithm is thus in most cases unusable for real-world size problems. The encouraging results on NP-hard problems make GAs a candidate for solving the grouping problems.

The rest of the paper is organized as follows. In Sections 2, 3, and 4 we present three ways that have so far been explored in tackling the grouping problems, each time pointing out what we feel are the weaknesses of those approaches. Addressing those drawbacks, we introduce the Grouping GA in Section 5. Section 6 illustrates the paradigm with a detailed description of its application to two different grouping problems, the Bin Packing and the Economies of Scale problems. An experimental comparison of the GGA with other GAs is given in Section 7. Conclusions are drawn in Section 8.

2. Standard GA Operators and Grouping Problems

In this section we examine the effects of the classic genetic operators on the structures relevant to the grouping problems. Application of a straightforward encoding scheme together with standard genetic operators (Holland, 1975) is the first route that has been taken in the GA literature treating grouping problems, for example, by Van Driessche and Piessens (1992), Ding, El-Keib, and Smith (1992), and Jones and Beltramo (1991). We will show why we think this is not the best GA approach for these problems.

2.1 Encoding and Redundancy

Let's assume the most straightforward encoding scheme, namely one gene per object. For example, the chromosome ADBCEB would encode the solution where the first object is in the group A, the second in the group D, third in B, fourth in C, fifth in E, and sixth in the group B.

Among the six design principles for constructing useful representations (see, e.g., Radcliffe (1991)), one can find the Principle of Minimal Redundancy—each member of the

[1] There are many more grouping problems of great practical importance, but their description would be too lengthy for the purpose of this paper.

[2] This is why we talk about *grouping* problems, rather than *partitioning*—we thus emphasize the importance of the groups, rather than the "cuts."

search space (here the space of all valid groupings) should be represented by as few distinct chromosomes as possible (ideally, exactly one) to reduce the size of the space the GA has to search.

As Mühlenbein (1992) points out, given a solution with m groups, $m!$ distinct chromosomes can represent it, each corresponding to one assignment of the names (A, B, . . .) to the groups. Thus the degree of redundancy (i.e., the number of distinct chromosomes encoding the same solution of the original problem) of this scheme grows more than *exponentially* with the number of groups, i.e., with the size of the problem.[3] Consequently, the size of the space the GA has to search is much larger than the original space of solutions and the power of the GA is seriously impaired.

2.2 The Context Insensitivity of Crossover

In grouping problems, the straightforward encoding leads to the highly undesirable effect of casting context-dependent information out of context under the standard crossover. For instance, in the chromosome ABCADD, the C associated with the third gene only has sense in the context of that particular chromosome, where it means that the third object is not grouped with any other object. Taking that C out of the context during crossover has disastrous consequences. To see this, let us cross this individual using the standard two-point crossover:

A | BC | ADD crossed with
C | AD | CBB would yield
CBCCBB as one of the two children.

In the absence of mutation, a recombination of two identical individuals should produce progeny that is again identical to the parents. The two parents above are identical with respect to the problem being solved by the GA because they both encode the same solution of the problem. Hence, an adequate recombination operator should produce an individual that again encodes the same solution. However, the resulting child above encodes a "solution" that has nothing in common with the solution its parents encode: there are two groups instead of four![4]

In other words, while the schemata are well transmitted with respect to the *chromosomes* under the standard encoding/crossover, their meaning with respect to the *problem* to solve (i.e., the cost function to optimize) is lost in the process of recombination.

2.3 Mutation

Let's again consider the standard encoding and see the effects of the standard mutation operator, i.e., a random modification of a randomly chosen allele. Consider, for example, the following chromosome: ABDBAC. A mutation of this individual could yield ABDEAC, which could be beneficial,[4] for the allele E, perhaps missing from the population, now appears in the genetic pool. The trouble begins as the algorithm approaches a good solution, developing large groups of identical alleles. The standard mutation of AAABBB would lead, for example, to AACBBB.

On one hand, the allele C appears in the population—a possibly beneficial effect. On the other hand, the new chromosome contains a "group" of just one element. Since grouping

[3] Note that in the problem treated by Mühlenbein (see Section 4, below), the number of groups, hence the degree of redundancy of the encoding, is constant. However, that is not the case for grouping problems in general.

[4] Depending on the problem's hard constraints, the resulting individual might be valid or invalid.

of objects usually accounts for a gain, this mutated individual will most probably show a steep loss of fitness in comparison with the other nonmutated individuals. Consequently, this individual will be eliminated with high probability from the population on the very next step of the algorithm, yielding hardly any benefit for the genetic search. In other words, the standard mutation operator is often too destructive once the GA begins to reach a good solution of the grouping problem.

3. Ordering GA Operators and Grouping Problems

In this section we examine the effects of *ordering* genetic operators on the structures relevant to grouping problems. Application of an encoding scheme representing *permutations* of the members of the set, together with ordering genetic operators, is a second route that has been taken in the GA literature treating grouping problems, for example, by Smith (1985), Bhuyan, Raghavan, and Elayavalli (1991), Jones and Beltramo (1991), and Reeves (1993). We will again show why we think this approach is not the best one for these problems.

3.1 Encoding and Redundancy

The basic idea of this approach is to represent permutations of the objects (members of the set **U** in Section 1, above) and use a decoding mechanism that reveals the actual assignment of the objects to groups corresponding to each chromosome. The decoding mechanism proceeds by considering the objects one by one in the *order* given by the chromosome and assigning them to the first group available.

For the sake of clarity, let's consider the Bin Packing Problem (BPP; see Section 6.2.1) with 10 objects to pack, numbered 0 through 9. Suppose that the objects are partitioned into three bins as follows:

0123 | 45678 | 9.

Any permutation of the objects leading to the same bin contents, such as

3210 | 45678 | 9

or

87645 | 1032 | 9

encodes the same solution of the original BPP. As for the straightforward encoding in Section 2, the degree of redundancy of this encoding grows exponentially with the size of the instance.

3.2 The Context Insensitivity of Crossover

Like the standard crossover operators in Section 2, most ordering crossover operators working with the permutation encoding cast context-dependent information *out of context* during recombination.

Indeed, given the mechanism of decoding the chromosome, it is clear that the meaning of a gene in the solution that the chromosome encodes depends heavily on all the genes that precede it on the chromosome and, in some cases, a few that immediately follow it.

To see this, consider Goldberg's (1989) PMX, probably the most widely used ordering crossover, and the instance of the Bin Packing Problem where objects of sizes

$1, 2, 3, 4, 6, 7, \ldots$, have to be packed inside bins of size 10. The PMX transmits the absolute positions of genes (o-schemata of type *a*) coming from one parent, and the relative positions of genes (o-schemata of type *r*) coming from the other. So let us see the meaning of the type *a* o-schema

??**23**??...

inside a few of the chromosomes that could inherit it:[5]

Chromosome/solution	*First two bins*	*Rest to pack*	*Group(s) containing the schema*
14**32** | 6 | 7...	10,6	7	1,2,3,4
17 | **32** | 64...	8,5	10	2,3
16**3** | **2**7 | 4...	10,9	4	1, 3, 6 and 2, 7 (schema broken)
4 | 7**3** | **2**16...	4,10	9	3, 7 and 1, 2, 6,... (schema broken)

The diversity of the sample is remarkable. There is no apparent *regularity* in the sample, be it with respect to the composition of the group(s) that contain the objects in the schema, or the composition of the other groups. Consequently, since it is precisely the composition of the groups[6] that is taken into account in the function being optimized, there is little insight that could be gained from the sample with respect to the expected quality of the solutions contained in the schema. In other words, sampling o-schemata is very unreliable[7] in BPP, and grouping problems in general. That is a serious drawback because the very basis of the GA, expressed by the Schema Theorem of Holland (1975), depends on a reasonably reliable sampling of schemata.

3.3 Mutation

The mutation operator of an ordering GA modifies the order of the genes on the chromosome. However, given the above, such an operator has a high probability of either not having any effect at all because of the high redundancy of the encoding or being too destructive because of its impact on most of the groups formed with the objects that map onto genes following the modified site.

4. Specialized Operators

Besides the approaches from the previous two sections, attempts have been made to better capture the structure of a grouping problem by Mühlenbein (1992) and Von Laszewski (1991). They use the standard encoding from Section 2, but use "intelligent" crossover operators in tackling the problem of *k*-way Graph Partitioning.

They realized that for a crossover to make sense with respect to the partitioning problem, it must transmit whole groups (partitions of a graph in their problem), rather than separated objects (nodes). The problems pointed out in Section 2 are handled as follows. When a partition A is to be copied from one parent into another, the most *similar*[8] partition B in

[5] For the sake of clarity, we show here the *sizes* of the objects, instead of their numbers.

[6] That is, the respective contents of the bins in the BPP.

[7] A sample of the corresponding o-schema of type *r* would be even more diverse, as it would include not only all the solutions above, but also those that have *any* number of objects in front of or between the two objects in the schema. The situation is even worse for o-schemata of order one: *any* solution could have *any* of them. Needless to say, it is precisely the short schemata that achieve the highest survival rates.

[8] Similarity is defined as the number of nodes the two partitions have in common.

the receiving parent is selected and the partitions in the receiving parent are renumbered so that B has the same number as A.[9] A is then copied. Since the problem requires partitions of equal size, the previous alleles of the loci in A\B[10] are distributed over the loci in B\A, to restore the constraint.

While this approach is certainly the one that went the farthest in the direction of the algorithm presented in the next section, we think it did not go far enough. For even if a crossover operator is well adapted to a problem, it cannot alone draw the maximum benefit from the GA technique, i.e., if it is not supported by an encoding adapted to the problem. Indeed, Mühlenbein does state that "The genetic representation [he uses] does not capture the structure of the problem."

One of the central points of the GA paradigm, emphasized already by Holland (1975), is the ability of a crossover to transmit *coadapted* sets of genes to offspring. These are the highly fit high-order schemata a GA is meant to develop from lower-order ones and maintain in the population. Indeed, the whole problem of schema disruption appears when a crossover is unable to correctly transmit the high-order schemata.

Mühlenbein and Von Laszewski specify how one group can be transmitted to a child. Von Laszewski explicitly limits crossover to the transmission of a single partition. Mühlenbein does say the crossover transmits complete partitions, but then also gives an example of insertion of a single one. In any case, while the operators they propose could certainly be extended to transmit several partitions, the choice of the partitions to be transmitted simultaneously remains unclear. In fact, the simple encoding they use does not offer any good answers to this question.

On one hand, when the genes corresponding to nodes belonging to different partitions of the graph are uniformly distributed over the whole chromosome, any substring cut out from that chromosome will contain members of many (possibly all) partitions of the graph.[11] Transmitting all of those partitions according to the mechanism above would lead to a (near-)complete destruction of the contents of the individual receiving those partitions, which would reduce the crossover to a mere mutation. It thus seems impossible to proceed in the usual way of copying the whole of a randomly selected crossing section from one parent into another.

On the other hand, copying just one partition does not take into account the fact that partitions can be coadapted, i.e., of poor quality when taken separately but efficient when present together in a chromosome. In the k-way Graph Partitioning Problem, this situation can arise when two partitions are strongly connected[12] but, as a couple, are almost isolated from the other partitions. Such partitions should be transmitted simultaneously.

One solution to this dilemma would be to select for transmission a small subset of the partitions present in an individual. But how should the subset be chosen? On one hand, selecting the partitions at random would be equivalent to assuming that any partition is equally well coadapted with any other. Although such a position is implicitly taken by researchers advocating uniform crossover, for example, Syswerda (1989), there is no doubt that it is the most disruptive one. On the other hand, trying to compute the most coadapted partitions prior to the crossover would actually amount to solving most of the partitioning problem outside (without) the GA.

In conclusion, the only way out of the dilemma seems to have the coadapted partitions

[9] Under the encoding scheme casting nodes as genes, this means that the alleles of all loci corresponding to nodes in B are set to the same value as those in A.

[10] A\B is the set of loci that are members of A in the first parent and *not* members of B in the second parent.

[11] Unless the crossover operator is restricted to transmit only extremely short substrings of chromosomes.

[12] That is, there is a relatively high number of edges connecting the two partitions.

represented in short group-schemata inside the chromosome. That leads us to the encoding proposed in the next section.

5. The Grouping Genetic Algorithm

The Grouping Genetic Algorithm (GGA) differs from a standard GA in two respects. First, a special encoding scheme is used to make the relevant structures of grouping problems become genes in chromosomes, i.e., the building blocks with which the GA works. Second, given the encoding, special genetic operators are used, suitable for the chromosomes.

5.1 The Encoding

As we have seen, neither the standard nor the ordering genetic operators are suitable for grouping problems. The reason is that the structure of the simple chromosomes (which the above operators work with) is *object*-oriented, instead of being *group*-oriented. In short, the above encodings are not adapted to the cost function to optimize. Indeed, the cost function of a grouping problem depends on the groups, but there is no structural counterpart for them in the chromosomes above. Note that these remarks are nothing more than a call for compliance with the Thesis of Good Building Blocks, central to the GA paradigm.

To remedy the above problems, we have chosen the following encoding scheme: the standard chromosome from Section 2 is augmented with a *group part*, encoding the groups on a *one gene for one group* basis.

More concretely, let us consider the first chromosome from Section 2 and the Bin Packing Problem. Numbering the objects from 0 through 5, the *object* part of the chromosome can be explicitly written

```
0 1 2 3 4 5
A D B C E B : ...
```

meaning the object 0 is in the bin labeled (named) A, 1 in the bin D, 2 and 5 in B, 3 in C, and 4 in E. This is the straightforward chromosome from Section 2. The *group* part of the chromosome represents only the groups (bins in BPP), e.g.,

```
... : B E C D A.
```

By a lookup in the object part, we can establish what the group names stand for, namely

B={2,5}, E={4}, C={3}, D={1} and A={0}.

In fact, the chromosome could also be written as

{2,5}{4}{3}{1}{0}.

The important point is that the genetic operators will *work with the group part* of the chromosomes, the standard object part of the chromosomes merely serving to identify which objects actually form which group. Note in particular that this implies that the operators will have to handle chromosomes of variable length.

In short, the encoding scheme we adopted makes the genes represent the groups. The rationale is that in grouping problems it is the groups that are the meaningful building blocks, i.e., the smallest piece of a solution that can convey information on the expected quality of the solution of which they are a part. This is crucial: indeed, the very idea behind the GA paradigm is to perform an exploration of the search space, so that promising regions are identified, together with an exploitation of the information thus gathered, by an increased search effort in those regions. If, on the contrary, the encoding scheme does not allow the building blocks to be exploited (i.e., transmitted from parents to offspring, thus allowing a continuous search in their surroundings) and *simultaneously* serve as estimators of quality of the regions of the search space they occupy, then the GA strategy inevitably fails and the algorithm performs, in fact, little more than a random search or naïve evolution.

5.2 The Crossover Operator

As pointed out in the previous section, a GGA crossover works with variable length chromosomes with genes representing the groups.

Given the fact that the hard constraints and the cost function vary among different grouping problems, the ways groups can be combined without producing invalid or too bad individuals are not the same for all those problems. Thus the crossover used will *not* be the same for all of them. However, it will fit the following pattern:

1. Select at random two crossing sites, delimiting the crossing section, in each of the two parents.
2. Inject the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that the crossover works with the group part of the chromosome, so this means injecting some of the groups from the first parent into the second.
3. Eliminate all objects now occurring twice from the groups they were members of in the second parent, so that the "old" membership of these objects gives way to the membership specified by the "new" injected groups. Consequently, some of the "old" groups coming from the second parent are altered: they do not contain all the same objects anymore, since some of those objects had to be eliminated.
4. If necessary, adapt the resulting groups, according to the hard constraints and the cost function to optimize. At this stage, local problem-dependent heuristics can be applied.
5. Apply the points 2 through 4 to the two parents with their roles reversed in order to generate the second child.

We give below more detailed examples of GGA crossovers adapted to two different grouping problems.

5.3 The Mutation Operator

A mutation operator for grouping problems must work with groups rather than objects. As for the crossover, the implementation details of the operator depend on the particular grouping problem on hand. Nevertheless, two general strategies can be outlined: either create a new group or eliminate an existing group. For some problems (given the hard constraints), a third one can be used: shuffle a small number of randomly selected objects among their respective groups. As with the crossover operator, more detailed descriptions of two GGA mutations are given below.

5.4 The Inversion Operator

The inversion operator serves to shorten good schemata in order to facilitate their transmission from parents to offspring, thus ensuring an increased rate of sampling of the above-average ones (Holland, 1975). In a Grouping GA, it is applied to the group part of the chromosome. Thus for instance, the chromosome

ADBCEB : BECDA

could be inverted into

ADBCEB : CEBDA.

Note that the object part of the chromosome stays unchanged. Indeed, the groups are still composed of the same objects.

The example illustrates the utility of this operator: should B and D be promising genes (i.e., well-performing groups), the probability of transmitting both of them during the next crossover is improved after the inversion, since they are now closer together, i.e., safer against disruption. That in turn makes the proliferation of the good group-schemata easier.

6. Two GGA Applications

In this section, we describe two concrete grouping problems to which the Grouping GA has been successfully applied. In both of them, the encoding scheme outlined in Section 5.1 above and the inversion operator from Section 5.4 have been used. We will thus concentrate on the details (i.e., the actual implementation of the strategies outlined in Sections 5.2 and 5.3, above) of the crossover and mutation operators, after a definition of each of the problems. We also report the experimental results obtained with those algorithms.

6.1 The Overall Procedure

In both applications described below, we used the following steady-state order-based GA procedure.

1. Generate at random an initial population of POPSIZE individuals.
2. Evaluate each individual in the current population according to the objective function to optimize. If the stop condition is attained, terminate.
3. Perform a noisy sort of the individuals according to the values obtained in step 2 (tournament of size 2 is used).
4. Take the first (best[13]) N_C individuals, $N_C \leq$ POPSIZE/2, and cross them over, replacing the N_C last (worst) individuals with the progeny.
5. Mutate N_M individuals selected at random from the current population.
6. Apply inversion to N_I individuals selected at random from the current population. At this point, one generation of the GA has been completed, i.e., a new population has been created.
7. Go to step 2.

[13] Given the noisy sort used in step 3, the first N_C individuals are about the best ones.

The parameters used were POPSIZE = 49 and 47, respectively, $N_C = 12$, $N_M = 4$, $N_I = 4$.

6.2 Bin Packing

6.2.1 The Problem Definition The Bin Packing Problem (BPP) is defined as follows (Garey & Johnson, 1979): given a finite set O of numbers (the object sizes), C (the bin's capacity), and N (the number of bins), is it possible to "pack" all the objects into N bins, i.e., does there exist a partition of O into N or less subsets, such that the sum of elements in any of the subsets doesn't exceed C?

This NP-complete decision problem naturally gives rise to the associated NP-hard optimization problem, the subject of this section: what is the *best* packing, i.e., what is the *minimum* number of subsets in the above mentioned partition?

Being NP-hard, there is no known optimal algorithm for BPP running in polynomial time. However, Garey and Johnson cite simple heuristics that can be shown to be no worse (but also no better) than a rather small multiplying factor above the optimal number of bins. The idea is straightforward: starting with one empty bin, take the objects one by one, and for each of them first search the bins so far used for a space large enough to accommodate it. If such a bin can be found, put the object there; if not, request a new bin. Putting the object into the first available bin found yields the First Fit (FF) heuristic. Searching for the most filled bin still having enough space for the object yields the Best Fit, a seemingly better heuristic, which can, however, be shown to perform as well (as badly) as the FF, while being slower.

Note that the problem treated by Smith (1985), although called there Bin Packing, differs from the BPP defined above in two respects. First, Smith considers *two-dimensional* objects, whereas we consider the *one-dimensional* problem of Garey and Johnson (1979). Second, and more important, Smith tries to pack as much as possible of the volume of the objects into one "bin" (actually a rectangular area), whereas we seek to pack all the objects into as few bins as possible. Clearly, these are very different problems. Indeed, Figure 1, where Δ is an arbitrarily small positive value, shows that there are instances of the BPP where the optimal solution contains only sub-optimally packed bins, even though at least one optimally packed bin can be found. Hence, a method using the algorithm of Smith will find the optimal bin and miss the BPP optimum.

Let's define a suitable cost function for the BPP. The objective being to find the minimum number of bins required, the first cost function that comes to mind is simply the number of bins used to "pack" all the objects. This is correct from a strictly mathematical point of view, but it is unusable in practice. Indeed, such a cost function leads to an extremely unfriendly landscape of the search space: a very small number of optimal points in the space are lost in an exponential number of points where this purported cost function is just one unit above the optimum. Worse, all those slightly suboptimal points yield the same cost. The trouble is that such a cost function lacks any capacity of guiding an algorithm in the search, making the problem a "needle in a haystack."

We thus settled for the following cost function for the BPP, due to Falkenauer and Delchambre (1992): maximize

$$f_{\mathrm{BPP}} = \frac{\sum_{i=1\ldots N} (F_i/C)^k}{N}$$

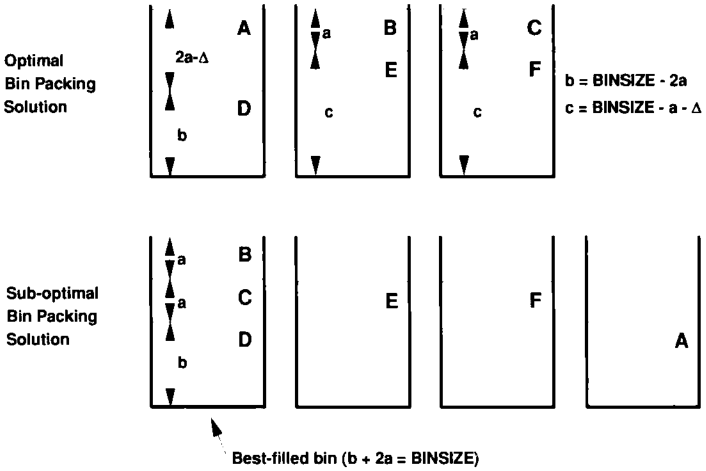


Figure 1. Best-filled Bin is not Bin Packing.

with N being the number of bins actually used in the solution being evaluated,[14] F_i the sum of sizes of the objects in (the fill of) the bin i, C the bin capacity, k a constant, $k > 1$.

The constant k expresses our concentration on the "extremist" bins in comparison to the less filled ones. The larger k is, the more we prefer well-filled "elite" groups as opposed to a collection of about equally filled bins. In fact, the value of k gives us the possibility to vary the "ruggedness" of the function to optimize, from the "needle in a haystack" ($k = 1$, $f_{\mathrm{BPP}} = 1/N$) up to the "best-filled bin" ($k \rightarrow \infty, f_{\mathrm{BPP}} \rightarrow \max_i [(F_i/C)^k]$).

We have experimented with several values of k and found out that $k = 2$ gives good results. Larger values of k seem to lead to premature convergence of the algorithm, as the local optima, due to a few well-filled bins, are too hard to escape.

6.2.2 BPCX—The Bin Packing Crossover Operator A crossover's job consists of producing offspring out of two parents in such a way that the children inherit as much as possible of the meaningful information from both parents. Since it is the bin[15] that conveys important information in BPP, we must find a way to transmit bins from the parents onto the children. This is done as follows.

Consider the following group parts of the chromosomes (recall that there is one gene per bin):

```
ABCDEF   (first parent)
abcd     (second parent).
```

First, copies are made of the two parents (in order not to destroy the parents) and two crossing sites are chosen at random in each of them, yielding, for example,

A | BCD | EF

[14] Recall that the optimization problem being solved here aims at minimizing N, i.e., N varies across various solutions.

[15] More precisely, a *coadapted subset* of the bins in the solution. See also Section 4.

and

`ab | cd | .`

Next, the bins between the crossing sites in the second chromosome are injected into the first, at the first crossing site, yielding

`AcdBCDEF.`

Now some of the objects appear twice in the solution and must be thus eliminated. Suppose the objects injected with the bins c and d also appear in the bins C, E, and F. We eliminate those bins, leaving

`AcdBD.`

With the elimination of those three bins we have, however, most probably eliminated objects that were not injected with the bins c and d. Those objects are thus missing from the solution. To fix this last problem, we apply the First Fit Descending (FFD) heuristic to reinsert them. That is, the missing objects are sorted in the inverse order of their sizes and then put one by one into the first sufficiently empty bin. This yields, say,

AcdBDx,

where x are one or more bins formed of the reinserted objects. As can easily be seen, the child just constructed indeed inherited important information from both parents, namely the bins A, B and D from the first and c and d from the second.

6.2.3 The Mutation Operator The mutation operator for the BPP is very simple. Given a chromosome, we select at random a few bins (i.e., groups) and eliminate them. The objects that composed those bins are thus missing from the solution and we use the First Fit (FF) heuristic to insert them back in a random order.

In order to improve the chances of the mutation to improve the current solution, we follow two rules: the emptiest bin is always among the eliminated ones, and we always eliminate and subsequently reinsert at least three bins (the number of used bins cannot be improved with less).

6.2.4 The Initial Population The initial population in a GA optimization is usually generated by a random sampling of the search space. However, it is a good rule to avoid introducing unacceptably bad individuals beside individuals of reasonable quality. Indeed, while the former individuals might contain valuable genetic material, their poor performance would make them quickly lose in competition with the latter. Not only would the information in the bad individuals be precluded from entering the rest of the population, but the (much) stronger individuals would quickly overtake the whole population, heightening the danger of premature convergence.

In short, the initial population should be random in the sense that large portions of the search space should be sampled, yet nonrandom in the sense of not too heterogeneous in terms of quality of the individuals as it is measured by the cost function.

Fortunately, this is relatively easy to achieve in the case of BPP: we generate the initial population by applying the First Fit heuristic to objects presented in random order. This yields a population of individuals containing a large variety of bins (i.e., genes) because the random order of objects in the FF heuristic leads to bins composed of different objects, yet none of the solutions is unacceptably bad because the heuristic creates solutions of reasonable quality.

6.2.5 Experimental Results Since the FFD can be shown to be a good heuristic for the BPP, it constituted a benchmark in tests of performance of our GGA approach.

We constructed the test data as follows: we first generated objects admitting a perfect packing (i.e., $f_{\mathrm{BPP}} = 1$), and then subtracted from randomly chosen objects a total of p percent of the size of one bin. For example, with $p = 10\%$ and the bin capacity $C = 255$, the total size of the objects was 25.5 less than the total capacity of the bins in the perfect packing. Thus, the test reflected the ability of the algorithm to get as close as possible to the optimum, rather than its eventual ability to find the optimum perfect packing. Indeed, the latter test would be a test of optimality, which would be equivalent to asking whether we can solve in a reasonable time an NP-complete decision problem—something the algorithm wasn't and couldn't be designed to do.

Since the total size of the objects was only a fraction of the bin size less than the total capacity of the bins in the perfect packing, the optimum number of bins, given by the perfect packing, hasn't changed. Hence we observed the ability of the algorithm to reach that number of bins, compared to the FFD heuristic. However, in order to account for a practical use of the GGA, we required it to reach the optimum number of bins in at most 5,000 generations, yielding running times of the order of one minute on a 4D35 Silicon Graphics (33 MIPS) workstation.

As for the size of the objects, we found that simply generating objects of sizes uniformly distributed between 0 and C leads to too easy instances: the inevitable small objects are easily "moved around" to fill up any remaining space in bins filled with the larger objects, and the GGA always finds the minimum number of bins in a short time. We therefore generated the object sizes in such a way that at most one object per bin in the perfect packing was smaller than a quarter of the size of the bin. Thus, for instances of 64 objects the average number of bins in the perfect packing (i.e., the optimum number of bins) was $N = 23$.

The results are summarized in Figure 2 for instances of 64 objects. The figure shows, as a function of p, the proportion of test cases successfully optimized by the FFD and the GGA with $k = 2$. To compute the proportions, 50 test cases were generated by the above procedure for each selected value of p (1.5 through 15% of C).

The figure shows the net superiority of the GGA in "tough" conditions, i.e., when the space for the objects to pack is tight.

One could object that the FFD heuristic is not a fair benchmark for assessment of merits of the GGA approach, since much better algorithms for the Bin Packing Problem are available, e.g., the Reduction Method of Martello and Toth (1990). However, a look at the crossover and mutation operators presented above reveals that the FFD and the simpler FF heuristics are the only domain knowledge or expertise exploited. Thus *all* of the observed difference of performance between the FFD and the GGA is really due to the GGA itself.

From this point of view and taking the position that it is the recombination power of the crossover that is the heart of a genetic algorithm, even more interesting is the plot showing the performance of the algorithm without the crossover operator (GNE, standing for Grouping Naive Evolution, in Figure 2). The data were obtained by running the algorithm on the same instances (and starting from the same initial populations) as for the GGA, but with the crossover replaced by reproduction: instead of crossing two parents to obtain two children,

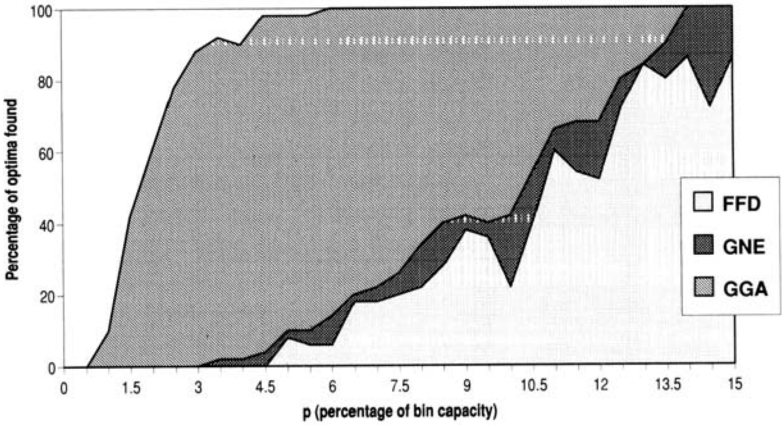


Figure 2. Relative performance of the FFD heuristic, the Grouping Naive Evolution (GNE), and the GGA.

the two parents were simply copied into the two children. Thus the information contained in the two parents was propagated into the population at the same rate as with the crossover, but it was not combined by the crossover, yielding a naive evolution algorithm.

The similarity of performance of the FFD heuristic and the naive evolution algorithm, on one hand, and the boost of performance when the BPCX crossover is enabled on the other, suggests that the crossover operator correctly combines building blocks meaningful to the Bin Packing Problem.

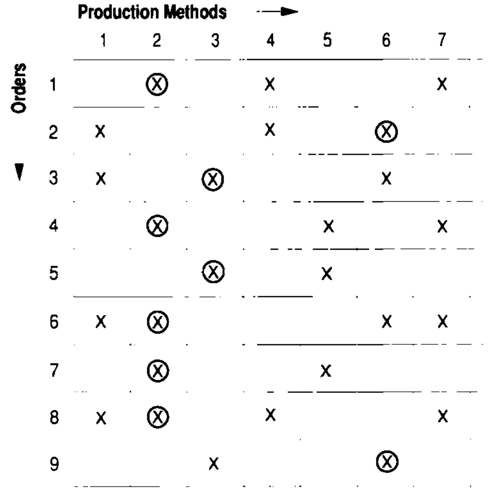
Concerning a comparison with more sophisticated methods, let us finally note that results of combining the above GGA with a local optimization inspired by the Reduction Method of Martello and Toth, reported by Falkenauer (1994), shows that such a hybrid approach yields an algorithm more powerful than either of its components. Given the prominence of the Reduction Method, that is a significant result for the GA in general and the GGA in particular.

6.3 The Economies of Scale

6.3.1 The Problem Definition The Economies of Scale Problem (ESP) is defined as follows (Falkenauer, 1991a). Given a list of orders, for each of them a list of possibilities of executing it, and a cost function expressing the cost of producing each order in each of the possible ways, select the way of producing each of the orders to minimize the total cost of the production.

More formally, we consider the following optimization problem: Consider a set O of objects O_i for which there is a set of possible attributes $A_{i,j}$. Select for each object in O exactly one attribute from A_i in order to minimize a cost function f.

The problem becomes nontrivial when the cost function decreases with *economies of scale*, i.e., when grouping of several orders into batches produced in the same way accounts for a gain. Indeed, when the cost of switching from one production method to the other and/or producing small quantities of goods in a uniform way grows, the fixed costs of producing



	1	2	3	4	5	6	7
1		⊗		X			X
2	X			X		⊗	
3	X		⊗			X	
4		⊗			X		X
5			⊗		X		
6	X	⊗				X	X
7		⊗			X		
8	X	⊗		X			X
9			X			⊗	

Figure 3. The Economics of Scale Problem.

each of the orders separately lose their importance in comparison with the gains attained by a mass production.

Figure 3 illustrates the problem. In that figure, each cross indicates a production method (a column) that can be selected for the corresponding order (the row). The circled crosses, representing the methods actually selected, represent a valid solution of the problem, since there is exactly one circle per row. Note the large number of circles in the second column: these orders are performed in the same way (in one batch), implying a big economy. The figure also illustrates the complexity of the problem: note that not all crosses in the third column were selected (circled). This seemingly implies a higher cost of the production, unless we realize that doing so would leave the cross in the second row isolated, yielding a very small batch and incurring an even higher overall cost.

The general form of the cost function to optimize is

$$f = f_S + f_G$$

f_S is the *static* part: this is the intrinsic cost of each order/method couple (a cross) taken in isolation. In particular, for a given order and production method, f_S will be the same whatever the length of the list of orders.

f_G is the *grouping* part: this term expresses the variation of the total cost of production due to an eventual grouping of orders into batches of orders produced according to the same production method. Typically, this term expresses the economies of scale one can achieve by reducing the number and/or increasing the size of the batches produced. For instance, the former arises when switching production methods is costly, and the latter when raw materials for each method are purchased for a price that decreases with the volume purchased.

With f_S constant over all crosses, f defines a purely grouping problem: only grouping of the orders into batches can improve the cost of the whole production.

With $f_G = 0$, the solution of the problem would be trivial: the optimum production would consist of choosing the cheapest method for each of the orders.

6.3.2 GX—The Grouping Crossover The grouping crossover proceeds by first copying the first parent into the future child. Then some of the groups of the second parent are injected into this child. The result is a child having some of the groups of the second parent and, as long as they were not destroyed by the injection, the groups of the first parent.

To see the function of the GX in detail, consider the following group parts of two chromosomes:

ABCDEFG

and

abcdef.

Each of the genes (A, B, . . . , a, b, . . .) represents a group of orders inside a batch. For instance, should the first of these chromosomes represent the solution depicted in Figure 3, the genes could read

A:	Prod.Method 2,	Orders 1, 4, 6, 7, 8, . . .
B:	Prod.Method 3,	Orders 3, 5, . . .
C:	Prod.Method 6,	Orders 2, 9, . . .

The GX proceeds first by selecting at random two crossing sites in each of the chromosomes, say

AB | CD | EFG

and

a | bc | def.

To construct the first child, the segment of the second parent delimited by the crossing sites is injected into the copy of the first parent, yielding

AB | bc | CDEFG.

Since the cost function decreases with the size of the groups, the injection of the groups is followed by an expansion of the injected groups, i.e., the orders not injected with the new groups are scanned and converted, whenever possible, to one of the new (injected) groups. For example, suppose that the chromosome containing the three genes A, B, and C above is injected with the following genes:

b:	Prod.Method 5,	Orders b,. . .
c:	Prod.Method 6,	Orders c,. . .

Then the orders 3, 4, 5, and 7 would be converted to those new groups, yielding

A:	Prod.Method 2,	Orders 1,6,8,. . .
B:	Prod.Method 3,	Orders . . .
b:	Prod.Method 5,	Orders 4,5,7,b,. . .
c:	Prod.Method 6,	Orders 2,3,9,c,. . .

Eventual double occurrences of groups, or groups eliminated by the expansion, are then eliminated, which gives in the case of the example, say,

```
AbcEFG.
```

Note that this child indeed inherited important information from *both* parents, namely the groups b and c from the second and, supposing not all of their members were converted by the injection/expansion, the groups A, E, F, and G from the first parent. The second child is obtained in the same way, but with the roles of the two parents permuted.

6.3.3 Two Grouping Mutations The first operator, the *grouping mutation*, proceeds as follows: it selects at random one of the production methods (columns in Figure 3) and converts to that group (by the same mechanism as above) orders randomly selected from those having that production method in their list of possible methods. In other words, crosses randomly selected in a column are circled, clearing the old circle in each of the affected rows. A new group is thus created, larger than if a single order changed methods, hence more likely not to decrease too much the overall quality of the individual.

The second operator, the *eliminating mutation*, selects at random one of the groups present in a chromosome and then tries to eliminate it. This is done by scanning the objects in the group and converting each of them to another group. If the group is not necessary, i.e., if each of its members has at least two possible attributes, the group will be completely eliminated.

6.3.4 Experimental Results We first tried to solve the problem with a standard GA, but we got no significant results, as the GA stagnated on poor solutions. This led to the development of the GGA technique.

6.3.4.1 Artificial Testbed To test the algorithm, we have considered a set of 254 "orders," each of them having an average of 15 possibilities of being produced, selected at random out of a set of 80. In other words, there were 254 rows and 80 columns, with an average of 15 crosses in each row. However, the objects were evenly divided into 6 groups according to the columns 0 through 5: each object had exactly one cross in these columns.

The cost function in this case had $f_S = 0$ (a pure grouping problem) and f_G = number of groups. This constituted our testbed—clearly, the best grouping consisted of selecting all the crosses in the first six columns.

In a sample of 10 typical runs, the best individual in the initial (random) population of POPSIZE = 47 individuals contained crosses in 71 columns (i.e., the orders were divided into 71 groups), sensibly near to the maximum of 80. However, an average of 38.2 generations of the GGA were enough to select the first 6 columns as the best basis for the grouping of the set, eliminating the random noise.

6.3.4.2 Industrial Implementation We have applied the above algorithm to the following problem from the metalworking industry. A list of orders for a foundry is given and for each order a list of possible ways of producing the ordered quantity of the metal. By selecting an appropriate method for each of the orders, economies of scale can be achieved by grouping the orders into batches produced in the same way. This results in a smaller number of production cycles, each of them concerning a larger quantity of metal. The production of large quantities implies smaller inventory relative to the total production.

The interesting point of this application is that here the static costs of producing one order sometimes vary considerably. Hence, the algorithm must find the proper way between the grouping that reduces the grouping cost f_G (mainly the cost of the inventory) and simple local optimization that reduces the static cost f_S.

Evaluating the performance of the algorithm in this case is more difficult because the optima are not known in advance. To get an idea anyway, we have compared the GGA to a classic enumerative method, the A*, being aware of no algorithm specifically designed for that problem.

On relatively small sets of data (say 50 orders with 10 possibilities for each of them), the GGA always found the same or a better solution than the A*, while taking less time. Given the fact that on small problems the A* had good chances to come across the optimum, we conclude that the GGA did so as well.

On larger sets of data (150 orders) the use of the A* algorithm becomes difficult because it is hard to estimate the influence of eventual groupings of a large number of orders on the total cost of the whole production. With a weak estimator, the A* algorithm easily overlooks good solutions because the search space is too large to be searched nearly exhaustively. In any case, the GGA performed substantially better and faster on large problems. More importantly, it enabled the end user to optimize previously unmanageable productions.

Recently the implementation of the algorithm was enhanced to handle lists of orders of up to 1,000 orders (rows) and 127 production methods (columns). The complexity of finding a good grouping for such large instances is daunting and well beyond the human capabilities or those of an enumerative algorithm like A*. Still, the algorithm finds good solutions to those problems in reasonable running times (a couple of hours on a 4 MIPS workstation).

7. Experimental Comparison with Other GAs

Several applications of the GGA show its superiority over the standard and ordering GAs when applied to grouping problems. We have already mentioned the poor performance of the standard GA on the Economies of Scale Problem, but in order to eliminate any bias of preference, it is probably best to compare the GGA to results obtained by other researchers.

7.1 Bin Packing

Reeves (1993) has recently tackled the Bin Packing Problem with an ordering GA coupled with a decoder of the chromosomes (permutations of items) using the First Fit and Best Fit heuristics. He used the procedure of Falkenauer and Delchambre (1992), described in Section 6.2.5 above, for generating instances of known optima and various degrees of difficulty. The experimental results obtained with the ordering GA were inferior to the ones reported above[16] for the GGA.

However, perhaps more interesting than this comparison is the method Reeves uses to enhance the performance of the ordering GA. In a hybrid algorithm, he performs a *reduction*[17] of the problem: each time a sufficiently filled bin is found in a solution (i.e. an individual decoded with the FF or BF), the bin is "fixed" and the objects it contains are eliminated from *all* individuals in the population, yielding a smaller problem.

[16] The procedure of Reeves finds the optimum less often than we do in Section 6.2.5 above, on instances of 50 rather than 64 items. A rerun of the GGA on 50 items yielded a performance significantly superior to the one reported above.

[17] This "reduction" is not to be confused with the Reduction Method of Martello and Toth, the latter being based on a criterion of *dominance* of one bin over others.

Recall that the purpose of the GGA crossover operator is to transfer *groups* (bins in BPP) from parents to offspring. Supposing a bin remains consistently above-average,[18] this mechanism will eventually propagate the bin throughout the whole population. Clearly, at that point the bin will be "fixed" (because *all* individuals in the population will contain it), and could be removed from the problem.

Thus the reduction of Reeves does the same thing as the GGA's crossover operator: it propagates promising bins. The difference lies in the fact that the GGA continuously samples the solutions containing the promising bins, increasing the sampling rate according to the observed quality of complete solutions containing those bins, whereas Reeves' procedure eliminates the bins permanently, on the basis of observed quality of the bins taken separately (i.e., disregarding the quality of the solutions that contain them). This has two consequences.

First, the GGA always retains the possibility to find out eventually that the promising bin actually isn't a part of an optimal solution,[19] whereas the Reeves procedure cannot recover the items eliminated with a bin. Consequently, once a wrong bin has been eliminated by the reduction, the latter procedure will have to settle to a local optimum, the global one being unattainable.

Second, the GGA's implicit estimates of bin quality through evaluation of whole solutions ensure an adequate evaluation of merit of each bin with respect to the whole problem being solved, whereas the reduction of Reeves explicitly promotes single bins on the simple basis of their being well filled, which is equivalent to assuming that the Bin Packing Problem is a mere composition of several Best-filled Bin Problems (see Figure 1).

On one hand, the propagation of promising bins through the reduction mechanism is faster than the one performed by the GGA because it bypasses the relatively slow process of implicit evaluation of the bins. On the other hand, the evaluation of the bins performed by the GGA is more thorough, in comparison to the reduction that explicitly strives for local optima of the kind depicted in Figure 1.

Thus, the reduction can be seen as validating the GGA (namely its propagation of groups), albeit in a manner that will probably prevent it from finding the global optima of difficult instances, like the ones considered by Falkenauer (1994).

7.2 Stock Cutting

Hinterding and Juliff (1993) applied the GA to the problem of stock cutting, where a set of rods (items) is given, as well as a set of stock sizes. The objective is to cut all the rods from the sizes available in the stock, in such a way that waste is minimized.

Thus, the problem is to find the best grouping of the rods, each group being cut from one of the available stock sizes. The problem can also be thought of as a Bin Packing Problem where the bins are of various sizes.

The authors considered several encodings of the problem, including the permutated list of items (i.e., an ordering GA), but concluded that "the GA search proved to be only marginally better than a random search." They finally adopted an encoding and operators inspired by Falkenauer and Delchambre (1992), i.e., the GGA, which yielded very good results.

[18] That is, the individuals containing that bin consistently score better than those that do not contain it. Recall that under the GGA encoding, a group represents a *group-schema* of order one.

[19] That is, the items have to be distributed into *several* bins in any optimal solution.

7.3 Equal Piles

Jones and Beltramo (1991) consider the problem of equal piles, where a given set of items of various sizes has to be divided into a given number of "piles," in such a way that the "heights" of the piles[20] are as equal as possible. A comparison with the results in that paper is perhaps the most valuable, because the authors precisely describe the instance of the problem they considered. It features 34 items to be divided into 10 piles, admitting the optimal solution with all piles equal (i.e., the sum of height differences equal to 0).

The authors tested five crossover operators with the standard GA (encoding group-numbers), and four with the ordering GA (encoding permutations of items). The best of those nine techniques turned out to be an ordering GA with the PMX crossover operator and an unspecified greedy heuristic. That technique achieved an average best value of 171, obtained after roughly 40,000 fitness evaluations. The best value ever reached was 2, i.e., the optimum was never obtained within 100,000 fitness evaluations.

We have applied the GGA to the Equal Piles Problem and tested its performance on the same instance, starting the GGA with 30 different initial populations. We obtained the value of 2 in *all* runs, in an average of only 226 generations (corresponding to less than 6000 fitness evaluations). More importantly, the GGA found a solution of the optimum value of 0 in 90% of the runs, in an average of 608 generations (15,200 fitness evaluations). In fact, the GGA's performance would hardly be visible on the comparative chart of Jones and Beltramo (1991), as it would quickly become indistinguishable from the horizontal axis (the optimum value of 0).

The Equal Piles GGA is so far a rather crude application of the method. There is no doubt that the performance can still be greatly improved, for instance, by inclusion of a smart heuristic in phase 4 of the GGA crossover (thus making up for the unspecified greedy heuristic used by Jones and Beltramo), or by promoting diversity in the population of 50 individuals (thus making up for the population of 1000 used by Jones and Beltramo).

8. Conclusions

Despite the praised (and well-documented) robustness of the standard and ordering GAs, they do not fare well on grouping problems because of their inappropriateness to the structure of those problems. Nevertheless, we have shown that a new encoding can lead to efficient GA operators for these problems.

The successful applications described suggest that the Grouping GA paradigm holds a promise for many other important problems. For instance, Falkenauer (1991b) uses the GGA for Conjunctive Conceptual Clustering, and first results of work currently under way in our lab on the Workshop Layouting Problem (see Section 1) suggest that the GGA will outperform any other currently available method for that problem.

We also hope to have made a convincing case for the importance of adequate encoding (and, consequently, genetic operators) for a successful application of the GA paradigm.

Acknowledgments

The author thanks Heinz Mühlenbein, Kenneth De Jong, and an anonymous referee for suggestions that helped improve the presentation of the paper.

[20] That is, the sums of the sizes of the items in each pile.

References

Bhuyan, J. N., Raghavan, V. V., & Elayavalli, V. K. (1991). Genetic algorithm for clustering with an ordered representation. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 408–415). San Mateo, CA: Morgan Kaufmann.

Ding, H., El-Keib, A. A., & Smith, R. E. (1992). *Optimal clustering of power networks using genetic algorithms* (TCGA Report No. 92001). Tuscaloosa, AL: University of Alabama.

Falkenauer, E. (1991a). A genetic algorithm for grouping. In R. Gutiérrez & M. J. Valderrama (Eds.), *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis* (pp. 198–206). Granada, Spain: World Scientific.

Falkenauer, E. (1991b). *A genetic algorithm for conceptual clustering* (Technical Report FMS39). Brussels, Belgium: CRIF Industrial Automation.

Falkenauer, E., & Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE 1992 International Conference on Robotics and Automation (RA92)* (pp. 1186–1192). Los Alamitos, CA: IEE Computer Society Press.

Falkenauer, E. (1994). *Setting new limits in bin packing with a grouping GA using reduction* (Technical Report RO108). Brussels, Belgium: CRIF Industrial Automation.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability—A guide to the theory of NP-completeness.* San Francisco, CA: W. H. Freeman.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning.* Reading, MA: Addison-Wesley.

Grefenstette, J. J. (Ed.) (1985). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Hinterding, R., & Juliff, K. (1993). *A genetic algorithm for stock cutting: an exploration of mapping schemes* (Technical Report 24 COMP3) Melbourne: Victoria University of Technology.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Jones, D. R., & Beltramo, M. A. (1991). Solving partitioning problems with genetic algorithms. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 442–449). San Mateo, CA: Morgan Kaufmann.

Männer, R., & Manderick, B. (Eds.) (1992). *Parallel problem solving from nature, 2, Proceedings of the Second Conference on Parallel Problem Solving from Nature* (PPSN2). Amsterdam: North-Holland, Elsevier Science.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. In *Discrete applied mathematics*, vol. 22. Amsterdam: North-Holland, Elsevier Science.

Mühlenbein, H. (1992). Parallel genetic algorithms in combinatorial optimization. In O. Balci, R. Sharda, & S. A. Zenios (Eds.), *Computer science and operations research—new developments in their interfaces* (pp. 441–453). New York: Pergamon Press.

Radcliffe, N. J. (1991). Forma analysis and random respectful recombination. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 222–229). San Mateo, CA: Morgan Kaufmann.

Reeves, C. (1993). Hybrid genetic algorithms for bin-packing and related problems. Submitted to *Annals of Operations Research.*

Smith, D. (1985). Bin packing with adaptive search. In J. J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications* (pp. 202–207). Hillsdale, NJ: Lawrence Erlbaum Associates.

Syswerda, G. (1989). Uniform crossover in Genetic Algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9). San Mateo, CA: Morgan Kaufmann.

Van Driessche, R., & Piessens, R. (1992). Load balancing with genetic algorithms. In R. Männer & B. Manderick (Eds.), *Parallel problem solving from nature 2* (pp. 341–350). *Proceedings of the Second Conference on Parallel Problem Solving from Nature* (PPSN2). Amsterdam: North-Holland, Elsevier Science.

Von Laszewski, G. (1991). Intelligent structural operators for the *k*-way graph partitioning problem. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 45–52). San Mateo, CA: Morgan Kaufmann.